DOWNLOAD PDF EMBEDDED SYSTEM DEVELOPMENT PROCESS

Chapter 1 : Embedded Control Systems Design/The design process - Wikibooks, open books for an open w

The development process of an embedded systems mainly includes hardware design process and software design process. Unlike the design process of software on a typical platform, the embedded system design implies that both hardware and software are being designed similarly Although this isn't continuously the case, it is a truth for many.

Introduction[edit] This chapter describes the process of designing a new embedded control system, or improving an existing one. That is, how can an individual engineer, or a team of engineers and project managers, tackle the design in a systematic way. With respect to the traditional views on the design process, this chapter adds an extra point of viewâ€"the design context of each particular systemâ€"to increase insight in the various interactions between the traditional design steps, and their relative importance. The last section treats some cross-sectional issues in the design of large-scale systems. System specification , describing the requested behaviour of the system. Detailed design , resulting in a concrete structure of all modules from which the system must be made. Implementation , building and testing prototypes of the eventual system. Some cases such as this example pass all of these phases, but it is difficult to give a clear description of what phases each new design has to go through exactly, and of what the design team should do exactly in each phase. Some designs will spend more time in the functional design phase, others on the system specification, etc. The market demands to deliver ever more complex systems, with ever more customization, life-long maintenance support, and recycling and disassembly requirements at the end of the active life of the system, have resulted in design activities that can no longer follow any of the traditional, very structured design process models. Except maybe for domains such as aviation , where strict regulations and certifications are to be satisfied. Traditional design process models[edit] This section discusses some of the popular, traditional design process models: The Waterfall model is the most rigid one, suggesting to move to a phase only when its preceding phase is completed and perfected. Phases of development in the waterfall model are kept very separated, and there is no room for iteration or overlap. The V model has the same strict serial structure as the waterfall model, but it suggests that, before going to a more detailed design level, one should already test all the system features and properties that can be tested at the current level of design abstraction. The Incremental Model allows multiple iterations in some of the design phases, resulting in a multi-waterfall process. The Spiral Model is similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations called Spirals in this model. Model-Driven Engineering is an emerging design process, that improves on the V-Model by supporting the test phases at each design level by software models that simulate the system before real implementations exist already. Design contexts[edit] In practice, the design process of a particular system is not only determined by what phases to go through, in what order, but also by the particularities of the context in which the system is to be designed: Does one design the first prototype of an innovative system based on still partially immature technology? Does one design a system to participate in a contest? The following sections bring some structure into this issue of design contexts, by presenting a non-exhaustive set of typical contexts. From Scratch Environment[edit] In this sort of project the design team starts working from scratch so everything needs to be organized and designed. This is an disadvantage as well as an advantage: This product is not a result from previous models, but it is designed out of nothing. The used techniques, the basic principle, the looks, Figure 1 Design process in a from scratch environment A typical project in this environment goes on the cycle shown in figure 1. Starting from the "requirement definition", which can be defined by the design team or deduced from a market research, the process undergoes a level of more and more details. Another important step is the testing or "implementing" phase. Because the product is completely new, the system should be tested thoroughly. The feedback of error messages is quite crucial at this point. These prototypes are also a source of adjustments in the architectural or detailed design. In extreme situations even the requirement definitions need to be redeveloped, e. This whole of iterations makes the project very time-consuming. Adapting Environment[edit] Here the design team starts from a previous model, version, We report the fact that there exist different

adaptations. Sometimes the system is expanded with totally new features, this can vary from one or two radical changes in design requirements to a lot of small adjustments. Another adaptation is the optimization of existing design requirements. In automotive industry, designers are usually working in this kind of environment. A car manufacturer typically has a few models which are improved with latest technologies e. Even when a new model in a new segment is developed, the design team can reuse older concepts or components like a chassis. Figure 2 Design process in an adapting environment Figure 2 shows there is obviously less feedback or iteration involved compared to a design process in a from scratch environment. The requirement definitions are mainly derived from market research. Customers are asked for the satisfaction and shortcomings of the current product so the company can counter these imperfections. In the automotive industry for example, concept cars are firstly introduced to the public. Depending on their reaction, the car manufacturer can decide whether a concept is ready for production, or is still in need of some modifications. In this case the market research results probably dealt with safety shortcomings. The engineers then have to translate this requirement into a new system. There was no customer that said: Prototypes can force the detailed design to change, because the required functions are not fulfilled, nevertheless iteration towards the system specifications is unusual. Nowadays the aspect of maintenance or after sale-support is becoming important. Total quality management is therefore a helpful tool. Competition Environment[edit] A lot of companies organize a competition where the winning team is accepted to produce or build the proposed product e. Typical for these kind of projects are the deadline and the strict boundary conditions or specifications which are dictated by the organizers. Often these targets are hard to succeed, and cause the designers to be creative and flexible. An example of this environment is the Robocup Challenge. In this competition different teams are trying to build a robot that is able to play a soccer game. Figure 3 Design process in a competition environment In figure 3 one can see the cycle that describes the design process. Although the concepts, used technologies and details can change very frequently the requirements are firm as a rock and irreplaceable symbolised by the stout frame in figure 3. These frequent conversions are only allowable if they are not too expensive. In some situations e. The main characteristic of these kind of projects is the flexible set-up of requirements. One starts from a given objective and by doing this, he finds related items where he can work on and which might lead to interesting new developments. Google for instance is famous for its "20 percent time". By defining the system specifications, by searching for new developments or by analysing the prototypes some interesting new topics may come up which replace or complete the earlier defined demands. Similar to the competition environment is the lack of the maintenance design step. Once there are clear results from a research project and the company decides to bring a specific product on the market, these steps become important and the project moves to a from scratch or adapting environment. A nice example of a project in a research environment is the Solar Impulse. A design team in a research environment focuses typically on one or two user requirements. Another important issue concerning this environment is money. Cross-sectional Issues[edit] The similarities or items that are significant in all design steps are discussed in this section. These cross-sectional issues also cover the continuity between the different design phases. Information on obvious cross-sectional issues, i. Late design commitments[edit] A good design will wait as long as possible to fill in the real components and hardware. This leaves open a lot more options for the implementation later on and thus gives more margin for unexpected situations. How far in the design process one can keep a broad view is often very market depending. The view can be kept broad until the point where a particular system or product is linked to a customer, this point is called the order penetration point. Depending on the product delivery strategy make to stock, assemble-to-order, make-to-order, engineer-to-order the view can be kept broad the whole process or is already determined by the end-user at the beginning of the design process.

Chapter 2 : IZITRON â€“ Embedded System Development

Embedded systems development requires the close collaboration and integration of skills from both hardware and software engineers to design a fully functioning system. A well planned and executed development process can make a significant impact on a development team's work efficiency, R&D.

What are the key steps you would be following each type you design a system. The key characteristic of embedded system is that it inherits much of its functionality from a well designed program. Everything the embedded system is able to do is through a program which is running inside the microcontroller. This program is a special type of "software" called a firmware. Because it is "firm" in nature because the embedded system once programed and deployed to the end user will be running the same program through out its life time. For example a TV remote control runs the same program which encodes key press data into serial bit stream and sends through an IR transmitter. Same is the case with a MCU inside a pen drive, digital watch and calculators. Your PC can run several other software in addition to these and also they can be replaced by their alternatives. So development of embedded software i. Step I â€" Development of Program A program is a step by step instruction given to any processor to perform certain task. Actually machines processor understands very low level commands known as opcodes this language is sometimes referred to as machine language. Each instruction is very low level and basic, and they are represented by numbers in memory. An AVR microcontroller has about such instructions to various types of operation. Since working with machine language is very difficult for humans. We have deviced many ways to generate the machine language instructions. They are described below. Assembler This is a piece of software running on the host not in the microcontroller. The assembled adds a layer over machine language that it takes a input program in assembly language. This assembly language is exactly the machine language but those numeric instructions are given nice names in English. For example LDS for load direct from data space. STS for store direct to data space. ADD for as the name say add contents of two registers. The assembler takes input the program file in assembly language and converts each of those instructions in their English names to a numeric values that the microcontroller can understand. Compilers of High level language Writing programs in assembly language is also difficult and very time taking process. It also require the user to have complete idea of processor architecture. Writing programs in such language is very easy. A program written in high level language cannot be directly executed on CPU! It must first be converted into machine language. Learning a programming language is NOT an one night affair! It takes years to get expert on them. Also some programming languages like C looks very tough to new programmers. Luckily I was born in the times when they taught programming to school kids like they teach MS Office nowadays! Compiler is a computer software that converts a program written high level language to machine code that can be directly executed on a CPU. Since compiler is a "product" thus it is developed by many companies and sold at different prices. I have selected a compiler that is also good for your pocket. I use avr-gcc for all my projects. Because a compiler takes as input the programs written in high level language in form of text files. You can theoretically create those program files in any text editor like Notepad under Windows. An IDE offers much more than just editing programs. Add and edit files to project. The editor is much more powerful than Notepad and is specially designed for writing programs in C and thus has following advantage. Your typing gets faster with the assistance of auto-complete where the editor offers you to chose long variable and function names from a list. Any syntax error is underlined RED in real time thus allowing to to make corrections. Better view of the program. Thanks to syntax highlight feature different parts of the program gets different colours and looks. IDE also manages the compiler settings which are very crucial for proper and error free compilation of the programs. It calls the compiler for your from just a click of menu. Shows the result of the compilation process. It also helps in debugging of the programs. Atmel Studio 6 Integrated Development Environment Once a program is developed in C language using the Atmel Studio 6 and compiled to get the executable file. The next step is to transfer this file from your development PC to the microcontroller chip. This HEX file can be found at the Debug folder inside your project folder. The extension of this file is. This is a piece of hardware that can talk to both your PC and also the microcontroller.

This enables it to get what PC is saying and write those to the microcontroller chip. You can buy a programmer from our store. Its task is to read data from the hex file generated by a C compiler and transfer them to the programmer hardware connected on the USB port. Here at eXtreme Electronics we have developed a programmer software that is very easy to use by a beginner. You can download it from here. Screenshot of programmer software A Development Board The final important piece required is a development board. The development board is a hardware board that makes it easy to work with microcontrollers during the learning phase. A simple development board has the following vital parts. A socket for placing microcontroller. Power supply circuit that helps easy connection with a DC adapter. It converts 12V from adapter to a clean 5v and feed this to microcontroller. It also makes these 5volts available in male headers so that user can get 5v for their own use. For example to power other ICs or module they want to interface with microcontroller. This is like the heart for MCU. It provides beats that makes the MCU take steps. For accurate timing in you application you need a crystal oscillator. It provides a voltage and temperature independent clock source. Here the programmer is connected using a cable. They are available in male headers so that user can make connection to them very easily. Some people try to make this circuit on a breadboard or general purpose PCBs. But they often fail. Because these type of construction is error pron. So we recommend user not to try to learning programming and soldering at same time! We will create a new project, type code using the editor and finally compile the source program to get the executable hex file.

Chapter 3 : Development Process of Embedded Systems

Development Process in Embedded Software Development Over the past few years, the functional requirements of systems comprised of software have increased extensively, due to the advancement of various technologies used in devices.

However, they may also use some more specific tools: In circuit debuggers or emulators see next section. Utilities to add a checksum or CRC to a program, so the embedded system can check if the program is valid. For systems using digital signal processing , developers may use a math workbench to simulate the mathematics. System level modeling and simulation tools help designers to construct simulation models of a system with hardware components such as processors , memories , DMA , interfaces , buses and software behavior flow as a state diagram or flow diagram using configurable library blocks. Simulation is conducted to select right components by performing power vs. Typical reports that helps designer to make architecture decisions includes application latency, device throughput, device utilization, power consumption of the full system as well as device-level power consumption. A model-based development tool creates and simulate graphical data flow and UML state chart diagrams of components like digital filters, motor controllers, communication protocol decoding and multi-rate tasks. Custom compilers and linkers may be used to optimize specialized hardware. An embedded system may have its own special language or design tool, or add enhancements to an existing language such as Forth or Basic. Another alternative is to add a real-time operating system or embedded operating system Modeling and code generating tools often based on state machines Software tools can come from several sources: Software companies that specialize in the embedded market Ported from the GNU software development tools Sometimes, development tools for a personal computer can be used if the embedded processor is a close relative to a common PC processor As the complexity of embedded systems grows, higher level tools and operating systems are migrating into machinery where it makes sense. For example, cellphones , personal digital assistants and other consumer computers often need significant software that is purchased or provided by a person other than the manufacturer of the electronics. Embedded systems are commonly found in consumer, cooking, industrial, automotive, medical applications. Household appliances, such as microwave ovens, washing machines and dishwashers, include embedded systems to provide flexibility and efficiency. Debugging[edit] Embedded debugging may be performed at different levels, depending on the facilities available. The different metrics that characterize the different forms of embedded debugging are: From simplest to most sophisticated they can be roughly grouped into the following areas: Interactive resident debugging, using the simple shell provided by the embedded operating system e. Forth and Basic External debugging using logging or serial port output to trace operation using either a monitor in flash or using a debug server like the Remedy Debugger that even works for heterogeneous multicore systems. An in-circuit emulator ICE replaces the microprocessor with a simulated equivalent, providing full control over all aspects of the microprocessor. A complete emulator provides a simulation of all aspects of the hardware, allowing all of it to be controlled and modified, and allowing debugging on a normal PC. The downsides are expense and slow operation, in some cases up to times slower than the final system. This is used to debug hardware, firmware and software interactions across multiple FPGA with capabilities similar to a logic analyzer. Software-only debuggers have the benefit that they do not need any hardware modification but have to carefully control what they record in order to conserve time and storage space. The view of the code may be as HLL source-code , assembly code or mixture of both. Because an embedded system is often composed of a wide variety of elements, the debugging strategy may vary. For instance, debugging a software- and microprocessor- centric embedded system is different from debugging an embedded system where most of the processing is performed by peripherals DSP, FPGA, and co-processor. An increasing number of embedded systems today use more than one single processor core. A common problem with multi-core development is the proper synchronization of software execution. A graphical view is presented by a host PC tool, based on a recording of the system behavior. The trace recording can be performed in software, by the RTOS, or by special tracing hardware. RTOS tracing allows

developers to understand timing and performance issues of the software system and gives a good understanding of the high-level system behaviors. Reliability[edit] Embedded systems often reside in machines that are expected to run continuously for years without errors, and in some cases recover by themselves if an error occurs. Therefore, the software is usually developed and tested more carefully than that for personal computers, and unreliable mechanical moving parts such as disk drives, switches or buttons are avoided. Specific reliability issues may include: The system cannot safely be shut down for repair, or it is too inaccessible to repair. Examples include space systems, undersea cables, navigational beacons, bore-hole systems, and automobiles. The system must be kept running for safety reasons. Often backups are selected by an operator. Examples include aircraft navigation, reactor control systems, safety-critical chemical factory controls, train signals. The system will lose large amounts of money when shut down: Telephone switches, factory controls, bridge and elevator controls, funds transfer and market making, automated sales and service. A variety of techniques are used, sometimes in combination, to recover from errorsâ€”both software bugs such as memory leaks , and also soft errors in the hardware: This encapsulation keeps faults from propagating from one subsystem to another, improving reliability. This may also allow a subsystem to be automatically shut down and restarted on fault detection. Immunity Aware Programming High vs. For low-volume or prototype embedded systems, general purpose computers may be adapted by limiting the programs or by replacing the operating system with a real-time operating system. Embedded software architectures[edit] There are several different types of software architecture in common use. Simple control loop[edit] In this design, the software simply has a loop. The loop calls subroutines , each of which manages a part of the hardware or software. Hence it is called a simple control loop or control loop. Interrupt-controlled system[edit] Some embedded systems are predominantly controlled by interrupts. This means that tasks performed by the system are triggered by different kinds of events; an interrupt could be generated, for example, by a timer in a predefined frequency, or by a serial port controller receiving a byte. These kinds of systems are used if event handlers need low latency, and the event handlers are short and simple. Usually, these kinds of systems run a simple task in a main loop also, but this task is not very sensitive to unexpected delays. Sometimes the interrupt handler will add longer tasks to a queue structure. Later, after the interrupt handler has finished, these tasks are executed by the main loop. This method brings the system close to a multitasking kernel with discrete processes. Cooperative multitasking[edit] A nonpreemptive multitasking system is very similar to the simple control loop scheme, except that the loop is hidden in an API. The advantages and disadvantages are similar to that of the control loop, except that adding new software is easier, by simply writing a new task, or adding to the queue. Preemptive multitasking or multi-threading[edit] In this type of system, a low-level piece of code switches between tasks or threads based on a timer connected to an interrupt. This is the level at which the system is generally considered to have an "operating system" kernel. Depending on how much functionality is required, it introduces more or less of the complexities of managing multiple tasks running conceptually in parallel. As any code can potentially damage the data of another task except in larger systems using an MMU programs must be carefully designed and tested, and access to shared data must be controlled by some synchronization strategy, such as message queues , semaphores or a non-blocking synchronization scheme. Because of these complexities, it is common for organizations to use a real-time operating system RTOS , allowing the application programmers to concentrate on device functionality rather than operating system services, at least for large systems; smaller systems often cannot afford the overhead associated with a generic real-time system, due to limitations regarding memory size, performance, or battery life. The choice that an RTOS is required brings in its own issues, however, as the selection must be done prior to starting to the application development process. This timing forces developers to choose the embedded operating system for their device based upon current requirements and so restricts future options to a large extent. These trends are leading to the uptake of embedded middleware in addition to a real-time operating system. Microkernels and exokernels[edit] A microkernel is a logical step up from a real-time OS. The usual arrangement is that the operating system kernel allocates memory and switches the CPU to different threads of execution. User mode processes implement major functions such as file systems, network interfaces, etc. In general, microkernels succeed when the task switching and intertask communication is fast and fail when they are slow. Exokernels

communicate efficiently by normal subroutine calls. The hardware and all the software in the system are available to and extensible by application programmers. Monolithic kernels[edit] In this case, a relatively large kernel with sophisticated capabilities is adapted to suit an embedded environment. This gives programmers an environment similar to a desktop operating system like Linux or Microsoft Windows , and is therefore very productive for development; on the downside, it requires considerably more hardware resources, is often more expensive, and, because of the complexity of these kernels, can be less predictable and reliable. Common examples of embedded monolithic kernels are embedded Linux and Windows CE. Despite the increased cost in hardware, this type of embedded system is increasing in popularity, especially on the more powerful embedded devices such as wireless routers and GPS navigation systems. Here are some of the reasons: Ports to common embedded chip sets are available. They permit re-use of publicly available code for device drivers , web servers , firewalls , and other code. Development systems can start out with broad feature-sets, and then the distribution can be configured to exclude unneeded functionality, and save the expense of the memory that it would consume. Many engineers believe that running application code in user mode is more reliable and easier to debug, thus making the development process easier and the code more portable. Additional software components[edit] In addition to the core operating system, many embedded systems have additional upper-layer software components. If the embedded device has audio and video capabilities, then the appropriate drivers and codecs will be present in the system. In the case of the monolithic kernels, many of these software layers are included. In the RTOS category, the availability of the additional software components depends upon the commercial offering.

Chapter 4 : Compiling, Linking, and Locating - Programming Embedded Systems, 2nd Edition [Book]

Everything the embedded system is able to do is through a program which is running inside the microcontroller. This program is a special type of "software" called a firmware. Because it is "firm" in nature because the embedded system once programed and deployed to the end user will be running the same program through out its life time.

With Safari, you learn the way you learn best. Get unlimited access to videos, live online training, learning paths, books, tutorials, and more. Compiling, Linking, and Locating I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. The only thing that has really changed is that you need to have an understanding of the target hardware platform. Furthermore, each target hardware platform is uniqueâ€”for example, the method for communicating over a serial interface can vary from processor to processor and from platform to platform. We focus on the use of open source software tools in this edition of the book. Open source solutions are very popular and provide tough competition for their commercial counterparts. The Build Process When build tools run on the same system as the program they produce, they can make a lot of assumptions about the system. This is typically not the case in embedded software development, where the build tools run on a host computer that differs from the target hardware platform. There are a lot of things that software development tools can do automatically when the target platform is well defined. For example, if all of your programs will be executed on IBM-compatible PCs running Windows, your compiler can automateâ€”and, therefore, hide from your viewâ€”certain aspects of the software build process. Embedded software development tools, on the other hand, can rarely make assumptions about the target platform. Instead, the user must provide some of her own knowledge of the system to the tools by giving them more explicit instructions. The process of converting the source code representation of your embedded software into an executable binary image involves three distinct steps: Each of the source files must be compiled or assembled into an object file. All of the object files that result from the first step must be linked together to produce a single object file, called the relocatable program. Physical memory addresses must be assigned to the relative offsets within the relocatable program in a process called relocation. The result of the final step is a file containing an executable binary image that is ready to run on the embedded system. The embedded software development process just described is illustrated in Figure In this figure, the three steps are shown from top to bottom, with the tools that perform the steps shown in boxes that have rounded corners. Each of these development tools takes one or more files as input and produces a single output file. More specific information about these tools and the files they produce is provided in the sections that follow. The embedded software development process Each of the steps of the embedded software build process is a transformation performed by software running on a general-purpose computer. To distinguish this development computer usually a PC or Unix workstation from the target embedded system, it is referred to as the host computer. The compiler, assembler, linker, and locator run on a host computer rather than on the embedded system itself. Yet, these tools combine their efforts to produce an executable binary image that will execute properly only on the target embedded system. This split of responsibilities is shown in Figure These tools are extremely popular with embedded software developers because they are freely available even the source code is free and support many of the most popular embedded processors. We will use features of these specific tools as illustrations for the general concepts discussed. Once understood, these same basic concepts can be applied to any equivalent development tool. The manuals for all of the GNU software development tools can be found online at http: Compiling The job of a compiler is mainly to translate programs written in some human-readable language into an equivalent set of opcodes for a particular processor. Everything in this section applies equally to compilers and assemblers. Together these tools make up the first step of the embedded software build process. Of course, each processor has its own unique machine language, so you

need to choose a compiler that produces programs for your specific target processor. In the embedded systems case, this compiler almost always runs on the host computer. A compiler such as thisâ€”that runs on one computer platform and produces code for anotherâ€”is called a cross-compiler. The use of a cross-compiler is one of the defining features of embedded software development. The GNU C compiler gcc and assembler as can be configured as either native compilers or cross-compilers. These tools support an impressive set of host-target combinations. The gcc compiler will run on all common PC and Mac operating systems. Additional information about gcc can be found online at http: This is a specially formatted binary file that contains the set of instructions and data resulting from the language translation process. Although parts of this file contain executable code, the object file cannot be executed directly. In fact, the internal structure of an object file emphasizes the incompleteness of the larger program. The contents of an object file can be thought of as a very large, flexible data structure. Although many compilers particularly those that run on Unix platforms support standard object file formats such as COFF and ELF, some others produce object files only in proprietary formats. Most object files begin with a header that describes the sections that follow. Each of these sections contains one or more blocks of code or data that originated within the source file you created. However, the compiler has regrouped these blocks into related sections. For example, in gcc all of the code blocks are collected into a section called text, initialized global variables and their initial values into a section called data, and uninitialized global variables into a section called bss. There is also usually a symbol table somewhere in the object file that contains the names and locations of all the variables and functions referenced within the source file. Parts of this table may be incomplete, however, because not all of the variables and functions are always defined in the same file. These are the symbols that refer to variables and functions defined in other source files. And it is up to the linker to resolve such unresolved references. Linking All of the object files resulting from the compilation in step one must be combined. The object files themselves are individually incomplete, most notably in that some of the internal variable and function references have not yet been resolved. The job of the linker is to combine these object files and, in the process, to resolve all of the unresolved symbols. The output of the linker is a new object file that contains all of the code and data from the input object files and is in the same object file format. It does this by merging the text, data, and bss sections of the input files. When the linker is finished executing, all of the machine language code from all of the input object files will be in the text section of the new file, and all of the initialized and uninitialized variables will reside in the new data and bss sections, respectively. While the linker is in the process of merging the section contents, it is also on the lookout for unresolved symbols. For example, if one object file contains an unresolved reference to a variable named foo, and a variable with that same name is declared in one of the other object files, the linker will match them. The unresolved reference will be replaced with a reference to the actual variable. For example, if foo is located at offset 14 of the output data section, its entry in the symbol table will now contain that address. It is a command-line tool that takes the names of all the object files, and possibly libraries, to be linked as arguments. With embedded software, a special object file that contains the compiled startup code, which is covered later in this section, must also be included within this list. The GNU linker also has a scripting language that can be used to exercise tighter control over the object file that is output. If the same symbol is declared in more than one object file, the linker is unable to proceed. It will likely complain to the programmer by displaying an error message and exit. On the other hand, if a symbol reference remains unresolved after all of the object files have been merged, the linker will try to resolve the reference on its own. The reference might be to a function, such as memcpy, strlen, or malloc, that is part of the standard C library, so the linker will open each of the libraries described to it on the command line in the order provided and examine their symbol tables. If the linker thus discovers a function or variable with that name, the reference will be resolved by including the associated code and data sections within the output object file. Unfortunately, the standard library routines often require some changes before they can be used in an embedded program. One problem is that the standard libraries provided with most software development tool suites arrive only in object form. You only rarely have access to the library source code to make the necessary changes yourself. Thankfully, a company called Cygnus which is now part of Red Hat created a freeware version of the standard C library for use in embedded systems. This package is called newlib. You

need only download the source code for this library from the Web currently located at http: The library can then be linked with your embedded software to resolve any previously unresolved standard library calls. In other words, the program is complete except for one thing: The addresses of the symbols in the linking process are relative. In fact, if there is an operating system, the code and data of which it consists are most likely within the relocatable program too. The entire embedded applicationâ€”including the operating systemâ€”is frequently statically linked together and executed as a single binary image. Startup code One of the things that traditional software development tools do automatically is insert startup code: Each high-level language has its own set of expectations about the runtime environment. For example, programs written in C use a stack. Space for the stack has to be allocated before software written in C can be properly executed. That is just one of the responsibilities assigned to startup code for C programs. Most cross-compilers for embedded systems include an assembly language file called startup. The location and contents of this file are usually described in the documentation supplied with the compiler. Startup code for C programs usually consists of the following series of actions: Zero the uninitialized data area. Allocate space for and initialize the stack. Typically, the startup code will also include a few instructions after the call to main.

Chapter 5 : Vulcan Enterprises LLC

Embedded System Development Process (Aero,Rail,Automotive) (16 ratings) Course Ratings are calculated from individual students' ratings and a variety of other signals, like age of rating and reliability, to ensure that they reflect course quality fairly and accurately.

Code the Applications and Optimize The coding of an embedded system can be done by using the following programming languages. Optimizing the code Verify the Software on the Host System Compile and assemble the source code into object file Use a simulator to simulate the working of the system Verify the Software on the Target System Download the program using a programmer device Use an Emulator or on chip debugging tools to verify the software Install the Program in the Chip To install the developed code into a microcontroller needs the following two items A Programmer Hardware The hardware of an embedded system can communicate to both the microcontroller and the PC. This allows it to get what the personal computer is saying and write those to the microcontroller chip. A Development Board The final and most essential piece is a development board. This board makes it easy to work with microcontroller while throughout the learning phase. A simple hardware development board has some important features. It helps in connecting with a DC adapter. It alters 12V from an adapter to a 5v for an operation of a microcontroller. It also makes these 5volts accessible in male headers so that the operator can get 5v for their operation. For instance, to power the module you need to interface with a microcontroller. Crystal Oscillator The crystal oscillator is the heart of the microcontroller unit. For exact timing of your application, you require a crystal oscillator. It offers a temperature and voltage independent CLK source. Here the programmer is linked using a cable. They are existing in male headers so that user can make a construction to them very simply. Applications of Embedded Systems The application areas of an embedded systems include Consumer electronics, Office automation, Industrial automation, Biomedical Systems, Field Instrumentation, Telecommunications, Wireless technology, Computer networking, Security, and Finance. Applications of Embedded Systems Thus, this is all about the various steps in developing the embedded systems. We hope that you have got a better understanding of this concept. Furthermore, any doubts regarding this concept, or to implement any electrical and electronic projects , please give your valuable suggestions by commenting in the comment section below. Here is a question for you, what are the programming languages used in embedded systems?

Chapter 6 : Embedded Systems Development Process & Services

Embedded System Development Processes W. T. Tsai Department of Computer Science and Engineering Arizona State University Tempe, AZ [email_address] Slideshare uses cookies to improve functionality and performance, and to provide you with relevant advertising.

Overview[edit] Model-based design provides an efficient approach for establishing a common framework for communication throughout the design process while supporting the development cycle V-model. In model-based design of control systems, development is manifested in these four steps: The model-based design paradigm is significantly different from traditional design methodology. Rather than using complex structures and extensive software code, designers can use Model-based design to define plant models with advanced functional characteristics using continuous-time and discrete-time building blocks. These built models used with simulation tools can lead to rapid prototyping, software testing, and verification. Not only is the testing and verification process enhanced, but also, in some cases, hardware-in-the-loop simulation can be used with the new design paradigm to perform testing of dynamic effects on the system more quickly and much more efficiently than with traditional design methodology. History[edit] The dawn of the electrical age brought many innovative and advanced control systems. As early as the s two aspects of engineering, control theory and control systems, converged to make large-scale integrated systems possible. In those early days controls systems were commonly used in the industrial environment. Large process facilities started using process controllers for regulating continuous variables such as temperature, pressure, and flow rate. Electrical relays built into ladder-like networks were one of the first discrete control devices to automate an entire manufacturing process. Control systems gained momentum, primarily in the automotive and aerospace sectors. In the s and s the push to space generated interest in embedded control systems. Engineers constructed control systems such as engine control units and flight simulators, that could be part of the end product. By the end of the twentieth century, embedded control systems were ubiquitous, as even white goods such as washing machines and air conditioners contained complex and advanced control algorithms, making them much more "intelligent". In , the first computer-based controllers were introduced. These early programmable logic controllers PLC mimicked the operations of already available discrete control technologies that used the out-dated relay ladders. The advent of PC technology brought a drastic shift in the process and discrete control market. An off-the-shelf desktop loaded with adequate hardware and software can run an entire process unit, and execute complex and established PID algorithms or work as a Distributed Control System DCS. Model-based design steps[edit] The main steps in Model-based design approach are: Plant modeling can be data-driven or based on first principles. Data-driven plant modeling uses techniques such as System identification. With system identification, the plant model is identified by acquiring and processing raw data from a real-world system and choosing a mathematical algorithm with which to identify a mathematical model. Various kinds of analysis and simulations can be performed using the identified model before it is used to design a model-based controller. First-principles based modeling is based on creating a block diagram model that implements known differential-algebraic equations governing plant dynamics. A type of first-principles based modeling is physical modeling, where a model consists in connected blocks that represent the physical elements of the actual plant. Controller analysis and synthesis. The mathematical model conceived in step 1 is used to identify dynamic characteristics of the plant model. A controller can then be synthesized based on these characteristics. Offline simulation and real-time simulation. The time response of the dynamic system to complex, time-varying inputs is investigated. This is done by simulating a simple LTI Linear Time-Invariant model, or by simulating a non-linear model of the plant with the controller. Simulation allows specification, requirements, and modeling errors to be found immediately, rather than later in the design effort. Real-time simulation can be done by automatically generating code for the controller developed in step 2. This code can be deployed to a special real-time prototyping computer that can run the code and control the operation of the plant. If a plant prototype is not available, or testing on the prototype is dangerous or expensive, code can be automatically generated from the plant model. This code can be deployed to the

special real-time computer that can be connected to the target processor with running controller code. Thus a controller can be tested in real-time against a real-time plant model. Ideally this is done via automatic code generation from the controller developed in step 2. It is unlikely that the controller will work on the actual system as well as it did in simulation, so an iterative debugging process is carried out by analyzing results on the actual target and updating the controller model. Model-based design tools allow all these iterative steps to be performed in a unified visual environment. Disadvantages[edit] The disadvantages of Model-based design, are fairly well understood this late in development lifecycle, of the product and development. One major disadvantage is that the approach taken is a blanket or coverall approach to standard embedded and systems development. Often the time it takes to port between processors and ecosystems can outweigh the temporal value it offers in the simpler lab based implementations. Much of the compilation tool chain is closed source, and prone to fence post errors, and other such common compilation errors that are easily corrected in traditional systems engineering. Design and reuse patterns can lead to implementations of models that are not well suited to that task. Such as implementing a controller for a conveyor belt production facility that uses a thermal sensor, speed sensor, and current sensor. That model is generally not well suited for re-implementation in a motor controller etc. Though its very easy to port such a model over, and introduce all the software faults therein. While Model-based design has the ability to simulate test scenarios and interpret simulations well, in real world production environments, it is often not suitable. Over reliance on a given toolchain can lead to significant rework and possibly compromise entire engineering approaches. Advantages[edit] Some of the advantages Model-based design offers in comparison to the traditional approach are: Engineers can locate and correct errors early in system design, when the time and financial impact of system modification are minimized. Design reuse, for upgrades and for derivative systems with expanded capabilities, is facilitated. Because of the limitations of graphical tools, design engineers previously relied heavily on text-based programming and mathematical models. However, developing these models was time-consuming, and highly prone to error. In addition, debugging text-based programs is a tedious process, requiring much trial and error before a final fault-free model could be created, especially since mathematical models undergo unseen changes during the translation through the various design stages. Graphical modeling tools aim to improve these aspects of design. These tools provide a very generic and unified graphical modeling environment, and they reduce the complexity of model designs by breaking them into hierarchies of individual design blocks. Designers can thus achieve multiple levels of model fidelity by simply substituting one block element with another. Graphical models also help engineers to conceptualize the entire system and simplify the process of transporting the model from one stage to another in the design process. This was soon followed by tool like sim and Dymola , which allowed models to be composed of physical components like masses, springs, resistors, etc. These were later followed by many other tools.

Chapter 7 : Embedded Systems - Design & Development Process - Embedded Technology & Application

Module 1 will introduce the learner to the components of your embedded system software development process. This module will be a quick overview for many topics with detailed analysis to follow in later modules and courses.

For example, our house is full of embedded systems: A car has about embedded systems on board. They make a car safer, cost-effective, provide ease of operation and comfort of movement. The fuel injection system, braking system, transmission control system, dashboard devices, central locking and audio system - these all are automotive embedded systems. They heat or cool car seats, turn mirrors, rotate lights, control the movement of the wipers and doors glasses. In some vehicle models, they can even measure the tire pressure, show the route to the destination point and determine if the driver is tired or not. Let us also think about the areas of our life in which embedded systems play a key role. Military security systems and government management systems are based on a set of high-performance embedded systems. Embedded systems also control different processes at the space stations and satellites. Any modern machine tool and measuring device - it is also an embedded system. Many complex medical diagnostic systems are using embedded systems for analysis, results processing, etc. In other words, the embedded system - it is a small computer, that is integrated into the device which it controls. Despite all the clear benefits of such systems, there are several myths users connect with embedded systems development. These myths suggest that embedded systems are inconvenient and inapplicable for the general use. It is very difficult to change the embedded system configuration. Honestly, after the embedded system has been installed, changing of its configuration - both hardware and software - is quite a challenge. But if the system will be originally programmed to be able to support configuration and different settings, it will greatly simplify software updating processes in the future. These requirements are then converted into a detailed description of the technical characteristics of the future device. Specifications must contain a complete description of the device operation modes including a reaction for each input signal, data, and error processing algorithms. All items that describe the technical characteristics of the device, should be discussed together with the client. Clear program documentation can also help to accelerate the introduction of the software changes. Many believe that embedded system design and development documentation - is a number of short comments for a working version of the program. However, good documentation of the program is as important as the software itself. Embedded system can not be scaled. Yes, one can not scale small embedded systems to a higher level without complete replacing of the core architecture. The insurmountable limitation here is the base software platform. Otherwise, larger projects have the rich functionality and are designed with the possibility to scale. So, is it possible to start with a small configuration and scale it to a desired size, if necessary? Yes, but it requires a special level of architecture that provides comprehensive scalability, automation, and informatization. Today there are embedded systems companies that provide an effective software architecture solutions with the help of which embedded systems can be significantly scaled in all directions. Such approach makes it possible to gradually develop software solutions in line with business needs without changing the core technology. Therefore, the most important step in the development of embedded systems is the right choice of a software platform. There are hardware limitations due to the memory restrictions. This is a popular myth. What is it based on? While developing embedded systems the idea of minimizing storage costs has been dominated. Now multi-core technologies came to shake things up. Multi-core processors have already occupied a central place in the product lines of the leading suppliers of semiconductor products with chips that have from two to eight cores. Compared with conventional integrated circuits, they provide greater computing power thanks to a parallel processing. Multi-core processors offer better system organization, as well as work at lower clock speeds. The main difficulty with multi-core systems is that the developers need to go from a serial performance model to a parallel performance model, where all the processes are carried out simultaneously. The higher level of parallelism, developers reach, the better the performance of multi-core systems they get. The effective use of multi-core technology significantly improves the performance and scalability of the network equipment, control systems, video game platforms and a variety of other embedded applications. Only assembler can be

used for embedded system software development. Many developers of embedded systems use for the devices programming only assembly language. Yes, a well-written program in assembly language runs faster and uses less memory space than the program written in a high-level language. This is important aspect at a relatively low speed and limited memory. Plus assembly language provides developers with direct access to any hardware devices. But here, arises a risk of codes incompatibility. In other words, when we will need to use another type of device we will spend a lot of time trying to adapt our program to another system assembly instructions. On the other hand, high-level languages allow creating a source code, which can be portable. On the basis of this source code, executable machine code is generated. Programs in high level language have a higher level of abstraction. Therefore, the programmer can complete the project in less time comparing to the same project in assembly language. By the way, if the same previously developed code will be used for future projects, it will significantly increase programming efficiency. The best time saving result can be achieved if we will use both high-level language and assembler for the project development. The main part of the application will be written in C, and time-critical algorithm fragments - in assembler. Embedded systems are relatively isolated and therefore are protected from a wide range of threats. However, the age of connected devices, underscore the need to think about these issues more serious. Modern devices are often connected to corporate networks, clouds, or directly to the Internet. On the one hand, it increases the functionality and ease of use, and on the other, it makes the device more vulnerable to external influences. The most effective way to ensure a reasonable balance between functionality and security devices is to determine the security requirements before the software development. This should be done in the context of the overall system and operating environment including a network environment. Moreover, the earlier these processes will start, the more effective they will work. Embedded device must be designed in the way to make it impossible for a potential attacker to hack the system. At the same time security system must be able to cope with a wide range of threats - from network attacks to a physical threats. So here are the most popular myths about embedded system development. If you want to get more information or want to develop an embedded system for your unique business purposes, mail us and our expert will give you a free consultation on the theme you are interested in.

Chapter 8 : Embedded system - Wikipedia

The next one in basic Embedded Systems Software Development Tools is a linker. A linker is a computer program that combines one or more object code files and library files together in to executable program.

Chapter 9 : The Leader in Embedded Software Development Tools

The embedded systems developer unfamiliar with the secure development process should study proven high-assurance development standards that are used to certify critical embedded systems. Two noteworthy standards are DOB Level A (a standard for ensuring the safety of flight-critical systems in commercial aircraft) and ISO/IEC (Common.